5 制御構文

5.1 上から下へ行くだけじゃない

Java プログラムは基本的に上から下へ処理をしていきます．例えば，

```
System.out.println("これが最初に表示");
System.out.println("次にこれを表示");
System.out.println("最後にこれを表示");
```

というプログラムは，

```
これが最初に表示
次にこれを表示
最後にこれを表示
```

と出力されます．順番がくずれることはありませんし，どれかを出力しないと言うこともありません．

しかし，これだけしかできないとちょっと応用力が足りません．プログラムを書いていると，もっと複雑な動作をさせる必要が出てきます．このように，単に上から下へプログラムを動かすのではなく，様々な動きを書式を**制御構文**といいます．

主な制御構文には，

- 条件分岐
- 繰り返し処理

があります。

条件分岐とは、その名もずばり、条件によってプログラムで実行することを変化させることを言います．

たとえば、とある携帯電話会社でパケット通信料金を計算するプログラムを作ることを考えてみましょう．この会社のパケット通信料は以下のような料金システムになっていたとします．

1、1000 パケットまでは一律 2000 円

2、1000 パケット以上なら，1 パケットに付き 2 円

では，何パケット与えたらイクラになるでしょう，という計算を Java で実装しようとすると，「1000 パケット未満」か「1000 パケット以上」かで計算式を変更しなければ行けませんよね．

もし条件分岐を知らないと、間違えた料金請求をしてしまって会社は大損です．そんなプログラムを作った貴方も減俸間違いなし．そんな恥だらけの人生を送らないためにも、条件分岐は大切です。

二つ目の重要な制御構文はは繰り返し処理です。繰り返し処理もその名のとおり、同じ

処理を何度も繰り返すことを言います。

先ほどのパケット通信料計算プログラムを見事作り上げたとしましょう．このとき，あなたは顧客全員のパケット通信料を計算しなければ行けませんよね．そのとき，いちいち顧客全員のでデータをプログラムに入力するのは大変です．同じような動作を繰り返すときは，繰り返し処理を行うとずっとラクチンになるはずです．

というわけで、この章では条件分岐、繰り返し処理の二つの処理について学んでいきます。

5.2 条件分岐・・・if

5.2.1 基本的な条件分岐・・・if

まず最初に勉強する条件分岐は、if 文です。if 文、もう名前を聞いただけで何をするのか予想がつきそうですが、「もし～だったら・・・」という意味を持つ文です。

早速例を見てみましょうか。

ここでは，まず手始めにこれまでに，先ほど例に挙げた携帯電話のパケット代計算プログラムを作ってみましょうか．ただし，最初に作るのは簡単にパケット代が最低料金分を超えているかどうかだけを判断するプログラムです．しかし，パケット量が 1000 未満の場合は自動的に 2000 円になりましたよね．そこだけに着目しますよ．

```
int packet = 1500;
if(packet < 1000){
        System.out.println("パケット代は 2000 円です");
}
```

結果を見てみると・・・

```
パケット代は 2000 円です
```

となり，packet の値が 1000 を超えていなければ支払う料金は一律 2000 円になることがわかりました．安心ですね．試しに packet を 1500 などにしてみると，

```

```

何も表示されません．あらら．

まあ，でも少なくともあまり使っていない人の分だけは，パケット代の計算がうまくいっていることが分かりますね．これで携帯でほとんどネットを見ない人からも確実に 2000 円徴収することが出来ます．携帯電話会社は大もうけ．

というわけで，これが if 文の最も基本的な形です．基本構文は以下の通りです．

```
if(条件式){
        ①//条件が満たされた場合
}
```

これで，条件が満たされた場合のみ①が実行され，それ以外の時は①はなかったことにされます．

なお，条件式は第 3 章で勉強した条件式(==や不等号など)か，boolean 型の変数を入れることが出来ます．

```
boolean kuufuku = true;
```

```
if(flag){
        System.out.println("ご飯を食べよう");
}
```

これでおなかが減ったらすぐご飯が食べられます．おじいちゃんも安心．

5.2.2 その他の場合・・・if-else

さて，先ほどの例だとパケット数が 1000 以下の場合のみ正しい値段を導出できましたが，いくらなんでもこれでは困ってしまいます．携帯会社としてはヘビーユーザからもちゃんと料金を取り立てたいですよね．

そんな場合「パケット数が 1000 以下ならば 2000 円」以外に，「**それ以外**ならパケット代×2 円」という条件を加えなければいけません．

さて，どうすればいいでしょうか？

まず一つ目の案としてはこんなものがあるでしょう．

```
int packet = 500;
int price = 0;
if(packet < 1000){
        price = 2000;
}
if(packet > 1000){
        price = packet*2;
}
System.out.println("パケット代は"+price+"円です");
```

これはなかなか良いアイデアです．パケット代が 1000 円以下の場合，以上の場合で二回 if 文を使って price を計算しています．

しかし，条件式が単純ならこれでもいいですが，もっと複雑になるとこれだと面倒くさくなってきます．

例えば，「今おなかが減っていて，時間が 12 時で，財布の中身が 1000 円以上ならばご飯を食べるけど，そうでなければ食べない」というプログラムを考えると・・・

```
boolean onaka;
int hour;
int money;
//…
if(onaka && hour == 12 && money >= 1000){
        System.out.println("ご飯を食べるぞ");
}
if(何を書けばいいんだ！？){
        System.out.println("ご飯を食べないぞ");
}
```

逆の条件を考えるのが大変すぎて，わけが分からなくなってしまいます．

Java ではそんな皆さんの気を楽にするための素敵な機能がついてきています．それが，else です．

else は「**if の条件以外の場合**」ということを意味します．

先ほどのパケット代計算式を else を使って書き直して見ましょう．

```
int packet = 500;
int price = 0;
```

```
if(packet < 1000){
        price = 2000;
}
else{
        price = packet*2;
}
System.out.println("パケット代は"+price+"円です");
```

先ほど条件式を書いていた部分が else に置き換わっていますよね．こうすることで，「パケット量が 1000 未満では**なかった場合**」どうするかを書くことが出来ます．

if と else を使った場合の構文を示します．

```
if(条件式){
        ①//条件が満たされた場合
}
else{
        ②//条件が満たされなかった場合
}
```

これで条件が満たされた場合は①が実行され，満たされなかった場合は②が実行されることになります．

5.2.3　　複数の条件がある場合・・・if-else if-else

さて，パケット量が 1000 を超えた段階でパケット代をパケット量×2 にしていましたが，最近ではダブル定額というのも流行らしいですね．どれだけ使ってもいくら以上にはならないというヘビーユーザには嬉しいシステムです．ちなみに作者は携帯電話でネットをやらないのでパケット代はいつも最低額ですが．

それはさておき，せっかくダブル定額制度があるのですからダブル定額制度をプログラム上で表現してみましょう．

今回は

1、 1000 パケットまでは一律 2000 円

2、 1000 パケット以上なら，1 パケットに付き 2 円

に加えて，

3、 2000 パケット以上なら一律 4000 円

ということにします．

さて，今度は if と else だけでは対応するのが難しそうです．もし頑張って if と else だけで対応するとこんな感じになるでしょうか．

```
int packet = 500;
int price = 0;
if(packet < 1000){
        price = 2000; //①
}
else{
         if(packet > 2000){
                        price = 4000; //②
         }
```

```
                else{
                        price = packet*2; //③
                }
        }
        System.out.println("パケット代は"+price+"円です");
```

ここでは，二回 if 文を使っています.

① は 1000 パケット以下なら 2000 円

② は 2000 パケット以上なら 4000 円

③ はそれ以外なら 1 パケット 2 円

という計算をしていることになります.

確かに，これでも正しく動きますがちょっと分かりづらいですよね.

このようにいくつかの条件で分岐したい場合 else if をりようすると楽になります.

```
        int packet = 3000;
        int price = 0;
        if(packet < 1000){
                price = 2000;
        }
        else if(packet > 2000){
                price = 4000;
        }
        else{
                price = packet*2;
        }
        System.out.println("パケット代は"+price+"円です");
```

これを実行してみると・・・

```
パケット代は 4000 円です
```

3000 パケットつかっているにもかかわらず料金は 4000 円で打ち止めです．これでパケット代を気にせずに携帯でネットが出来るようになりましたね．筆者への応援メールも書き放題です．待ってます.

というわけで，結果は先ほどの else if を使わない場合と同じですが，見た目は大分すっきりしたのではないでしょうか？

else if は，「それ以外で，この条件を満たす場合は」という意味になります．これによって，複数の条件を書くことができるわけですね.

ちなみに，else if 条件式はいくつでも書くことが出来ます．if 文がスタートして，else が来るまでいくつでも付け加えることが出来ちゃうんです．うまく使うことで，もっと複雑なパケット代計算式だって対応できちゃいます.

「1000 パケットまでは一律 1000 円だけど．その後 1 パケット 2 円になる．でも，2000 パケットを超えたら 1 パケット 0.5 円にまけてあげるよ．そのうえ，5000 パケットを超えたら一律 1 万円だ！でも 10 万パケットを超えるような使い方をしたら 100 万円罰金ね. あ，そうそうぴったり 7777 パケットだったら特別タダにしてあげるよ，それから，今特別キャンペーンで・・・」

なんていう見ているだけでイライラしてくる料金システムでも対応可能です.

というわけで，if-else if-else の構文をご紹介します.

```
if(条件式①){
        ①//条件式①が満たされた場合
}
else if(条件式②){
        ②//条件式②が満たされた場合
}
else if(条件式③){
        ③//条件式③が満たされた場合
}
. . .
else{
        ④//条件が満たされなかった場合
}
```

- 条件式①が成り立てば、①を行う
- 条件式②が成り立てば、②を行う。
- 条件式③が成り立てば、③を行う。
- ・・・
- どの条件も成り立たなければ④を行う

以上が、if~else if～else 文の処理の流れになります。

ちなみに，if 文を書く場合は，if 文だけがあればよくて，else if や else 文は必須ではありません．したがって，else 文などは必要なければ書かなくてもかまいません.

5.3 複数条件分岐・・・switch

5.3.1 値によって変化させます・・・switch-case

さて、if 文では一つ一つの条件を if-else if-else でつないでいました。条件文には何を入れても大丈夫です．不等号を入れても，boolean 型の変数を入れても大丈夫です.

これに対して，ある整数型変数の値が特定の値かどうかで条件分岐する制御文があります．それが，switch 文です.

これに対して、switch 文はある変数の値が特定の値だったらこういう行動をする、ということを決めるために利用します。もちろん、if 文ですべて書くことも出来ますが、もうちょっとスマートな書き方といえるかもしれません。

まあ、口で説明しても難しいので実際に例を見てみましょう。

ここでは，1 分あたりの通話料が深夜と昼間と夜間でそれぞれ 5 円，10 円，15 円の契約をした携帯電話の通話料について考えて見ましょう．

我々が目指すプログラムは，時間帯を指定すると一分間の通話料がいくらか教えてくれるプログラムです．プログラムを動かせばこれから電話をしていくらかかるかが一目で分かります．超便利．

ところで，深夜とか昼間とかはどうやって表現しましょうか．String 型を使って

```
String time = "深夜";
```

とする方法もありますが，一般的には int 型を使って表現することが多いです．

例えば，

```
int time = 0;
```

として，0 ならば深夜，1 ならば昼間，2 ならば夜間ということにします．

さて，ここで time の値によって条件分岐して値段を決めるわけですが，前節でやった if 文を使って書くこともできます．気が向いた人はチャレンジしてみてください．意外と簡単だと思いますから．

ですが，ここでは新しいテクニック，すなわち switch-case を利用します．

料金判定システムは switch-case を利用すると以下のように書くことが出来ます．

```
int time = 0; //0:深夜 1:昼間 2:夜間

int price = 0;
switch(time){
case 0:
        price = 5;
        break;
case 1:
        price = 10;
        break;
case 2:
        price = 15;
        break;
}

System.out.println("通話料は 1 分で"+price+"円です");
```

これを実行してみましょうか．

```
通話料は 1 分で 5 円です
```

というわけで，time が 0 ならばちゃんと 1 分 5 円であることが分かりました．これなら彼女と 1 時間話したってたった 300 円しかかかりません．電話し放題です．

ちなみに，time の値を 1 にすると・・・

```
通話料は 1 分で 10 円です
```

となりました．昼間は電話を控えた方がいいかもしれません．

動きを見たら分かるとおり，time の値によって実行する部分が変化するのですが，それ

だけだと良く分からないかもしれませんのでもう少し詳しく説明したいと思います.

まず, switch 型を書くためには必要不可欠な 3 つの要素 **switch, case, break** について説明しましょう。

まずは、最初の **switch** です。これはどの変数の値を見るかを指定する部分です．ここで指定された変数の値によって、動作が変化するわけです。ただし、ここで注意。switch で使える変数は **int 型の変数と enum 型のみ**です。double はもちろん、boolean も long も使えませんので注意しましょう。(脚注：enum 型についてはいずれ説明しますので、ここでは int 型で使うものだと思ってください。)

switch の書き方は以下のとおりです.

```
switch(変数名)
```

次に、**case** です。これは、switch に書いてある変数の値が case のあとにかかれている値と同じだったら、その行から実行していくという意味です。今回の場合、time が 0 ならば、case 0:に飛ぶことになります。このように switch 文は対象となる変数の値によって、

```
case 値:
```

の行にプログラムを移動させます。ちなみに、このとき値には整数しか書けません。まあ,int 型は整数しか入れられないわけですから, 文字列などを書いたところでそれと一致することはありえないので，書くだけ無駄ですからね.

また、値のあとには必ずコロン(:)を書きます。これは，こういう決まりだと思ってください.

そして、最後の **break** は，条件分岐した文を実行する範囲を決めるものです．break とかかれていたら，その時点で処理が終了して，switch 文を抜けることになります.

```
break;
```

さて，これで break の説明は終了です．あら，簡単.

ちなみに, break は今後も使われることが多々ありますので名前だけでも覚えておいてあげてください.

以上を踏まえて先ほどのコードがどう動くのかを説明しましょう.

```
int time = 0; //0:深夜 1:昼間 2:夜間

int price = 0;
switch(time){ //①
case 0:   //②
          price = 5; //③
          break; //④
case 1:
          price = 10;
          break;
case 2:
          price = 15;
          break;
}//⑤

System.out.println("通話料は 1 分で"+price+"円です");
```

まず，switch 文が①で開始されます．このとき，条件分岐の基となる変数が time である

ことが示されます.

次に，time の値がいくつかによって移動する場所が変更されます．この場合は，深夜，つまり time=0 ですよね．0 の時は，case 0:に移動しますから，このプログラムで言うと②に移動します.

その後，③が実行されます．これで，price=5 となりました.

そして，その次の行④で break があります．break と書いてあったら switc 文を抜けるという決まりがありますから，一気に⑤へ移動します．間にあった price=10 などは無視されますよ.

これによって，最終的に price の値は 5 になって，

```
通話料は 1 分で 5 円です
```

と表示されたのでした.

どうでしょうか？それほど難しくはないと思います.

さて,以上 switch 文をまとめると以下のようになります.

```
switch(変数名){
case 値 1:
          変数が値 1 だったときの処理
          break;
case 値 2:
          変数が値 2 だったときの処理
          break;
・・・
}
```

となります．ちなみに，switch 文ではいくつでも条件を加えられますので，case 文はいくつでも付加することが出来ます.

5.3.2 その他の場合・・・default

さて，先ほどのプログラムですが，time の値が 0 から 2 の間のときは case 文がありました．では，もし 3 以上の値だったりしたらどうでしょうか？

```
int time = 3; //0:深夜 1:昼間 2:夜間

int price = 0; //①
switch(time){
case 0:
          price = 5;
          break;
case 1:
          price = 10;
          break;
case 2:
          price = 15;
          break;
}

System.out.println("通話料は 1 分で"+price+"円です");
```

さあ，このありえない数値を入れた場合の処理です．深夜でも昼間でも夜間でもないん

ですから，いつなんでしょう？まあ，夕方かもしれませんが・・・いずれにせよ，想定外の値です.

さて，このときの一分間の通話料はいくらでしょうか.

通話料は 1 分で 0 円です

なんと，0 円になってしまいました.

これは驚きです．ただで好きなだけ通話できてしまうことになってしまいました．携帯会社は大損です.

なぜこんなことになったかといえば，プログラム中の①で price の値を初期化するときに 0 を与えているからなんですね.

switch 文は当てはまるものがなかった場合，何も実行せずに終了するという決まりがあります．したがって，price の値が初期値の 0 のままで終ってしまうのです.

これでは困ってしまいます．どうすればいいでしょうか？一つには，他の値が来た場合についても書いておくという手があります.

```
int time = 3; //0:深夜 1:昼間 2:夜間

int price = 0; //①
switch(time){
case 0:
          price = 5;
          break;
case 1:
          price = 10;
          break;
case 2:
          price = 15;
          break;
case 3: //③
          price = 10;
          break;
}

System.out.println("通話料は 1 分で"+price+"円です");
```

とりあえず，time の値が想定外の値である 3 だった場合については，通話料を昼間と同じ 10 円ということにしておきましょう.

通話料は 1 分で 10 円です

これで，なんとかなった気がします．しかし，皆さんはもうお気づきでしょうが，time の値が 3 だとは限りませんよね．4 かもしれませんし，10 かもしれません．本当はあってはならない値ですが，そういう値が来る可能性もあるわけです．そういう値に対して全ての条件を書くのはちょっとばかし大変です.

何しろ int 型は-2147483648 ～ 2147483647 まで扱えるのですから，いちいち全部書いていられません.

しかしご安心を．if には else があったように，switch にも**その他の場合の処理**を書く方法があります．それが **default** です.

default を使って先ほどのコードを直してみましょう.

```
int time = 3; //0:深夜 1:昼間 2:夜間

int price = 0;
switch(time){
case 0:
          price = 5;
          break;
case 1:
          price = 10;
          break;
case 2:
          price = 15;
          break;
default:
          System.out.println("想定外の値です. ");
          price = 10;
          break;
}

System.out.println("通話料は1分で"+price+"円です");
```

ここでは，price を 10 にするとともに，想定外の値が入ってきたことを知らせるメッセージもおまけで入れておきました.

さて，実行するとどうなるでしょうか？

```
想定外の値です.
通話料は1分で10円です
```

どうでしょうか. ちゃんと想定外の値が入ってきたことを知らせつつ，値段は一分 10 円ということになりました. これで，携帯電話会社は通話料をとりはぐる心配はありません.

さて，もうお分かりかと思いますが，default は，それまでに書いてあるすべての case に当てはまらなかった場合に実行される処理になります. if 文で言うところの else とまったく同じ動きをします.

だったら else と書いてくれれば良いのに，と思いますが，なぜか switch の場合は default と書く決まりがあるのです. そういうものだと思ってください.

なお，default も else と同じように，一番最後に書かなければいけないという決まりがあります. そこだけ注意してください.

さて，以上で switch の説明は終わりになります. 最後に switch-case-default 文の構文を紹介しましょう.

```
switch(変数){
case 値1
          対象変数が値1だったときの処理
          break;
case 値2:
          対象変数が値1だったときの処理
          break;
case 値3
          対象変数が値3だったときの処理
          break;
・・・
```

```
default:
          どのケースにも当てはまらなかった場合の処理
          break;
}
```

ちなみに、これを if 文で書きなおすと，

```
if(対象変数 == 値 1){
          対象変数が値 1 だったときの処理
}
else if(対象変数 == 値 2){
          対象変数が値 2 だったときの処理
else if(対象変数 == 値 3){
          対象変数が値 3 だったときの処理
}
・・・
else{
          どのケースにも当てはまらなかった場合の処理
}
```

となります。どちらで書いても実行したときの結果はまったく一緒です.

じゃあ,if 文と switch 文，どちらで書くのがよいのでしょうか？

一見まったく同じですが，switch 文では**ひとつの変数の値によって動作が変わる**という意味があります。もし、if 文で書いたら、途中で別の変数を条件分岐に利用することもできますが、switch 文の場合はそれができません。

というと if 分のほうが優れているような気がしますが、**必ずある変数を条件にしている**ということが保証されているため、プログラムを読みやすくなるという効果があります。ぱっと見ただけでプログラムを理解できるかどうかは、結構重要ですので switch 文も積極的に利用しましょう。

ただし、switch 文では、int 型しかつかえない上に、==の関係でしか使えませんので、やっぱり制限も多いんですけどね。

5.4 繰り返し処理～for～

5.4.1 for の基本

繰り返し処理というのは、その名の通り、同じ作業をなんども繰り返す処理のことです。プログラムを書いていると、同じ処理を何回も繰り返したくなるというのがあるんです。そんなとき、繰り返したい回数だけ同じ処理を書くのは面倒くさい上に、非効率的ですよね。というわけで、ここでは繰り返し処理について学んでいきましょう。

JAVA において繰り返し処理を行うためのキーワードは、**for**, **while**, **do-while** です。ここでは、まずもっとも基本的な繰り返し処理 **for 文**を学びましょう。この for 文は「初期条件を決めて、ある条件を満たすまで、繰り返し処理を行いながら繰り返す」というなんだか分かるような分からないような文です。

まずは、例を見るのが手っ取り早いでしょう。ここでは，毎日のパケット通信料を一週間分合計するプログラムを書いてみましょう.

そのためには 7 日分のデータを足さなければいけませんよね。

じゃあ、配列を使ってこんな風にしましょうか。

```
int[] packetPriceArray = {100,120,13,0,41,32,100};
int priceOfWeek = 0;

priceOfWeek += packetPriceArray[0];
priceOfWeek += packetPriceArray[1];
priceOfWeek += packetPriceArray[2];
priceOfWeek += packetPriceArray[3];
priceOfWeek += packetPriceArray[4];
priceOfWeek += packetPriceArray[5];
priceOfWeek += packetPriceArray[6];

System.out.println("今週のパケット代は"+priceOfWeek+"円です。");
```

ちょっぴり面倒くさいですね。それでも一応答えはちゃんとでます。

```
今週のパケット代は 406 円です。
```

でも、こんな面倒くさいこと 7 日分なら何とかかけますが、一か月分だったら同でしょう？あるいは、1 年分だったら・・・？気が遠くなります。

というわけで、同じことを繰り返す場合には for 文を使うのがだいぶ楽になります。

```
int[] packetPriceArray = {100,120,13,0,41,32,100};
int priceOfWeek = 0;

for(int i = 0; i < 7; i--){
          priceOfWeek += packetPriceArray[i];
}
System.out.println("今週のパケット代は"+priceOfWeek+"円です。");
```

さて、実行した結果どうなったでしょうか？

```
今週のパケット代は 406 円です。
```

先ほどと同様に 406 円使ったことが分かりました。

なんと、たった 3 行で先ほど 7 回繰り返した計算ができてしまいました．一生懸命 7 日分足すプログラムを書くより省エネです。

さて、こんな便利な for 文ここでは 7 回繰り返しただけですが、実際**は初期化をした後，ある条件を満たすまで、一定の行動を繰り返す**という意味の文です。ここでは、「i を 0 に初期化して，i が 5 未満の内は，1 ずつ i を増やす」意味になります。

for 文の構造はこうなっています。セミコロン;で区切って、三つの式を書くことになります。それぞれの式の意味は以下の通り。

```
for(初期化; 条件式; 繰り返し処理)
```

for 文に入ってきたプログラムは、まず**初期化**を行います。例文だと、int i=0 の部分ですね。これは、通常に書いた場合と同じで、変数 i に 0 を代入するという意味です。こんな()の中で書くなんてちょっと変な気がしますが、こういうこともできてしまうんです。

さて、次に出てくるのは、i<5 です。これは、**条件式**と呼ばれるものです。この条件式の値が、true である限り for 文後の{}の中身が繰り返されることになります。

そして、最後が**繰り返し処理**になります。この例でいうと、i++ですね。これは、{}の中

の処理が全部終わった後に実行される式です。この場合は、i の値が 1 増えます。以上が for 文の構文となります。

以上をまとめると、このような動きになります。

1. i=0 とする
2. i<7 が成り立つか確認
3. i<7 が成り立つなら｛｝内の処理を実行
4. i++を実行
5. 2 へ戻る

i の値が 7 になって**条件式を満たさなくなった場合、for 文は終了されます**。for 文直後の{}の直後までプログラムは一気に移動します。一週間分以上のデータを間違えて計算してしまうことはありません。

以上で for 文の説明を終わりです。なんて簡単。

って、これだけだと分かり辛いかもしれませんから、ちょっと変更してみましょうか。こんな感じで for 文を変更してみます。ここでは足すのと同時にいくら使ったか毎日表示しましょう，

```
int[] packetPriceArray = {100,120,13,0,41,32,100, 102};
int priceOfWeek = 0;

for(int i = 0; i < 8; i++){
          priceOfWeek += packetPriceArray[i];
}
System.out.println("今週のパケット代は"+priceOfWeek+"円です。");
```

これで、実行してみましょう。

```
今週のパケット代は 508 円です。
```

今度は８日分の合計金額が表示されるようになりました.

このように、条件式を変更することで繰り返し回数を変えられるわけです。

5.4.2 指定回数繰り返し処理

For 文のよくある使い方として、複数回まったく同じ行動を繰り返すことがあります。たとえば、下記のようなものがあります。

```
for(int i = 0; i < 10; i++){
          System.out.println("10 回数えるよ");
}
```

このようにすることによって 10 回同じことを繰り返すことができます。

```
10 回数えるよ
10 回数えるよ
10 回数えるよ
10 回数えるよ
10 回数えるよ
10 回数えるよ
10 回数えるよ
```

```
10回数えるよ
10回数えるよ
10回数えるよ
```

これによって、このようにまったく同じ動作を10回繰り返すことができます。

ところで、このような数え上げをする場合は、0からスタートさせて目標回数未満の判定をするのが一般的です．プログラムになれていない内は変な感じがするかも知れません．ですが，これはここお見せしたとおり、配列を利用する場合 0 からスタートして、配列の値-1まで数えるほうが便利だからなんです。

たとえば、毎月の携帯電話代を配列で表現することを考えて見ましょう。このとき、携帯電話には基本料金がかかるので、すべてのデータにあらかじめ基本料だけは入れておきたいと思ったとします。

この場合、以下のようにすることになります。

```
int[] priceArray = new int[12];
for(int i = 0 ; i < priceArray.length; i++){
        priceArray[i] = 2500;
}
```

これによって、配列の要素すべてが2500円になり、あらかじめ基本料が保存された配列を作成することができるようになります。

なお、具体的にはiが0からスタートし、まずpriceArray[0]に2500を代入します。同様に、配列の長さより 1 だけ小さい添え字の要素 priceArray[11]まで同じことを繰り返します。これによって配列のすべての要素に対して初期化をしてくれるということになります。

もちろん配列と無関係な繰り返し処理をするならこういう書き方をしてもかまいません。

```
for(int i = 1; i <= 10; i++){
        System.out.println("10回数えるよ");
}
```

ただ、10回繰り返して同じことをしたいだけならこれでも大丈夫です。

が、一流のプログラマを目指す皆さんは，是非0～最大値-1書き方でfor文を書くようにして下さい．

5.4.3 無限ループ

さあ、初期化と条件式を変更してみたのですから、最後は繰り返し処理を変更して見ましょう。繰り返し処理は以下のように変更してみましょうか。

```
for(int i = 0; i < 10; i--){
        System.out.println("10回数えるよ？");
}
```

今度は、i—によって、iの値を1ずつ減らしていっています。ん？減らす？ちょっとまって、減らしていったら、iは一生10より小さいんじゃ・・・？

なんだか不安ですが、とりあえず試してみましょう。

```
10回数えるよ？
```

```
10回数えるよ？
10回数えるよ？
10回数えるよ？
・・・
10回数えるよ？
10回数えるよ？
10回数えるよ？
10回数えるよ？
・・・
```

と無限に続いていきます。もう、一生プログラムは終わりません。終わらせたければctrl+Cを押して途中終了しましょう。

なぜこうなるのかは分かってもらえると思います． i が小さくなっていくんですから，そりゃ条件式は永久に満たされ続けるため，終わるわけがないですよね．このようなミスは結構ありがちですので，気をつけましょう．プログラムが終わらないと割とビックリです．

今回は、分かりやすいのでいいのですが、時にはうっかり条件式を一生満たしてしまうプログラムを書いてしまうことが結構よくあります。

こういった状態になることを**無限ループ**、あるいは、**暴走した**といいます。結構ありがちなプログラムミスなので、for文を書くときは注意しましょう。ちなみに、筆者は**今日も仕事中に無限ループなプログラムを作ってしまいました**。なんてタイムリー。

さて，以上でfor文の説明は終わりです．最後にfor文の構文を示しておきましょう．

```
for(初期化;条件式;繰り返し処理){
        処理;
}
```

また，この実行の流れはこうなります．

1. 初期化を行う
2. 条件式で判定
3. 条件式が満たされていなければ終了
4. 処理を行う
5. 繰り返し処理を行う
6. 2へ戻る

ポイントは，繰り返し処理が最後に行われることでしょうか．

5.5 条件を満たしている限り繰り返し・・・while、

5.5.1　　繰り返し条件が先・・・while

さあ、お次の繰り返し処理は、**while** 文です。実は、while 文は for 文よりはるかに簡単です。なにしろ，for 文を書くには，初期化と条件式と繰り返し処理を書かなければ行けなかったのに対して，while 文には**条件式**しかありません。おお、こりゃ簡単。では、早速例を見てみましょうか。

ここでは，一か月分のパケット利用料を使って、何日目に最低料金 2000 円を超えてしまうかを調べるプログラムを作成して見ましょう。

これを while 分を使って作ってみるとこうなります.

```
int[] packetPriceArray = {
                        100, 120, 13, 120, 123, 32, 66,
                        65, 0, 0, 543, 10, 25, 40,
                        100, 130, 200, 210, 31, 40, 10,
                        0, 50, 0, 15, 30, 10, 100
};

int totalPrice = 0;

int days = 0;
while(totalPrice < 2000){  //①
          totalPrice += packetPriceArray[days];
          days++;
}
System.out.println("2000 円を超えるのは"+days+"日後");
```

このプログラムを動かすとどうなるでしょうか？何日で3000円を超えてしまうのかドキドキです.

```
2000 円を超えるのは 23 日後
```

というわけで、23 日目には最低料金を超えてしまうことが分かりました。これ以上の利用すると使った分だけ料金が取られてしまいます。高校生のおこずかいにはちょっと負担になってくるかもしれません.

さて，このコード，ポイントは①の while 文のみです．while 文は条件式が成り立つ限り処理を続けるという文ですので，ここでは totalPrice が 2000 以下である限り処理が繰り返されるということになります.

では，2000 円を超えるまでの日数を数えるプログラムがどう動いているのか具体的に確かめるために，一部追加コードを加えましょう.

```
int[] packetPriceArray = {
                        100, 120, 13, 120, 123, 32, 66,
                        65, 0, 0, 543, 10, 25, 40,
                        100, 130, 200, 210, 31, 40, 10,
                        0, 50, 0, 15, 30, 10, 100
};

int totalPrice = 0;
```

```
int days = 0;
while(totalPrice < 2000){  //①
        totalPrice += packetPriceArray[days];
        days++;

        System.out.println("現在"+days+"日目．合計料金は"+totalPrice+"円");
}
System.out.println("2000円を超えるのは"+days+"日後");
```

while 文の繰り返し処理を行うたびに，今何日目でいくらかを表示するようにしました．さあ，この実行結果はどうなるでしょうか？

```
現在1日目．合計料金は100円
現在2日目．合計料金は220円
現在3日目．合計料金は233円
現在4日目．合計料金は353円
現在5日目．合計料金は476円
現在6日目．合計料金は508円
現在7日目．合計料金は574円
現在8日目．合計料金は639円
現在9日目．合計料金は639円
現在10日目．合計料金は639円
現在11日目．合計料金は1182円
現在12日目．合計料金は1192円
現在13日目．合計料金は1217円
現在14日目．合計料金は1257円
現在15日目．合計料金は1357円
現在16日目．合計料金は1487円
現在17日目．合計料金は1687円
現在18日目．合計料金は1897円
現在19日目．合計料金は1928円
現在20日目．合計料金は1968円
現在21日目．合計料金は1978円
現在22日目．合計料金は1978円
現在23日目．合計料金は2028円
2000円を超えるのは23日後
```

while 文の中を繰り返し行っていることが分かります．このとき，条件式が totalPrice<2000 ですので，price の値が 2000 未満である限り繰り返されます．出力を見ると，確かに料金が 2000 円を超えるまでは繰り返されていますよね．そして，料金が 2000 円になった 23 日目に処理は終了しています．

どうでしょうか？大体 while 文の構文は分かったのではないかと思います.

while 文は，ある条件が満たされる限りずっと同じ行動をするよということを表現したいときによく使われます.「電話に出ない限り，呼び出し音を鳴らし続ける」とか「借金を払うまでは地の果てまで追いかけていってやる！」なども while 文で書けるかも知れません.

最後に，while 文の構文は以下のとおりです.

```
while(条件式){
        //処理
}
```

for 文に比べると初期化と繰り返し処理が無くなっている分簡単ですね.

5.5.2 繰り返し条件が後・・・do-while

さて、さっきは while 文の説明をしましたが、ちょっと変形したものに **do-while** 文があります.

実は，do-while 文は while 文とほとんど変わりがありません．違いは，処理を一度必ず行うかどうかだけなんです.

では、具体的に先ほどの while 文を do-while で書き直してみましょう。

```
int[] packetPriceArray = {
                    100, 120, 13, 120, 123, 32, 66,
                    65, 0, 0, 543, 10, 25, 40,
                    100, 130, 200, 210, 31, 40, 10,
                    0, 50, 0, 15, 30, 10, 100
};

int totalPrice = 0;

int days = 0;
do{
        totalPrice += packetPriceArray[days];
        days++;

        System.out.println("現在"+days+"日目．合計料金は"+totalPrice+"円");
}while(totalPrice < 2000);
System.out.println("2000 円を超えるのは"+days+"日後");
```

さあ、while との違いは分かりますか？while 文は最初に while とかいてありましたが、今回は最初に do とだけ書いてあって、{} の最後に while と条件が書いてあり、最後にセミコロンがかいてあるのが違うだけです。さて、実行してみましょうか。

```
現在 1 日目．合計料金は 100 円
現在 2 日目．合計料金は 220 円
現在 3 日目．合計料金は 233 円
現在 4 日目．合計料金は 353 円
現在 5 日目．合計料金は 476 円
現在 6 日目．合計料金は 508 円
```

```
現在 7 日目．合計料金は 574 円
現在 8 日目．合計料金は 639 円
現在 9 日目．合計料金は 639 円
現在 10 日目．合計料金は 639 円
現在 11 日目．合計料金は 1182 円
現在 12 日目．合計料金は 1192 円
現在 13 日目．合計料金は 1217 円
現在 14 日目．合計料金は 1257 円
現在 15 日目．合計料金は 1357 円
現在 16 日目．合計料金は 1487 円
現在 17 日目．合計料金は 1687 円
現在 18 日目．合計料金は 1897 円
現在 19 日目．合計料金は 1928 円
現在 20 日目．合計料金は 1968 円
現在 21 日目．合計料金は 1978 円
現在 22 日目．合計料金は 1978 円
現在 23 日目．合計料金は 2028 円
2000 円を超えるのは 23 日後
```

実行した結果は、while 文の時とまったく同じでした。ようするに、do-while 文も while 文も動きはほとんど変わらないということです。だったら while 文だけでいいような気がしますよね。

では、do-while 文と while 文の決定的な違いはなんでしょうか。それを確認するために、先ほど while 文、do-while 文に以下のような変更を入れてみましょうか。

すなわち、最初からある程度パケットを使ってしまっている状態から、さらにパケットを使っているとします。

具体的には、totalPrice があらかじめ 2500 円だった場合を考えて見ましょう。

while 文の場合、以下のようにかけますね。

```
//前略
int totalPrice = 2500;

int days = 0;
while(totalPrice < 2000){
        totalPrice += packetPriceArray[days];
        days++;

        System.out.println("現在"+days+"日目．合計料金は"+totalPrice+"円");
}
System.out.println("2000 円を超えるのは"+days+"日後");
```

これで実行してみるとどうなるでしょうか。

```
2000 円を超えるのは 0 日後
```

おっと、いきなり終了です。0 日目で利用をやめてしまいました。while 文の場合は最初

から条件が満たされていなければ何もせずに終了してしまいます。

一方、do-while の場合はどうでしょうか？

while 文の場合、以下のようにかけますね。

```
//前略
int totalPrice = 2500;

int days = 0;
do{
        totalPrice += packetPriceArray[days];
        days++;

        System.out.println("現在"+days+"日目. 合計料金は"+totalPrice+"円");
}while(totalPrice < 2000);
System.out.println("2000 円を超えるのは"+days+"日後");
```

これを実行すると・・・

```
現在 1 日目. 合計料金は 2600 円
2000 円を超えるのは 1 日後
```

というわけで、すでに 2000 円を超えているにもかかわらず一日分利用してしまいました。

これが、while 文と do-while 文との違いです。すなわち、

- while 文は、条件に合わない場合は一度も実行せずに終了する可能性がある。
- do-while の場合**少なくとも一回は実行してから条件判定をする**ことになる。

というわけで、絶対に一回は実行したい場合は、do-while を使うようにしましょう。たとえば、while 分の中である値を別の変数に代入したい場合などは do-while 文を利用することになるでしょう。

では、最後に do-while 文の一般系をご紹介します。

```
do{
        繰り返し処理;
}while(条件式);
```

do-while の場合、while（条件式）の後にセミコロン;を必ずつけます。付け忘れるとコンパイルエラーが起きますよ。

逆に while 文のときはセミコロン;をつけてはいけませんよ。

5.5.3 強制的に条件へ戻る continue

さて、ここでは制御文の中でもちょっと特殊な二つをご紹介しましょう、それが、**continue と break** です。break は switch のところでも使いましたが、今回は使い方がちょっと違います。

break は中止、中断という意味がありますが、Java では、**ループを終了させる**効果があります。

一方、continue は続くという意味ですが、JAVA プログラミングでは、繰り返し処理の中で使われると、**強制的にループの先頭まで戻る**動作をします。

具体例を見ながら説明しましょう。

ここでは、例のごとくパケット使用量計算を利用します。

1、1000 パケットまでは一律 2000 円

2、1000 パケット以上なら，1 パケットに付き 2 円

という条件で、一ヶ月分のパケット使用量を計算することを考えて見ましょう。

```
int[] packetPriceArray = {
                    100,120,13,120,123,32,66,
                    65,0,0,543,10,25,40,
                    100,130,200,210,31,40,10,
                    0,50,0,15,30,10,100
};

int totalPrice = 0;
int packet = 0;

for(int i = 0; i < packetPriceArray.length; i++){//③
          packet += packetPriceArray[i];
          if(packet < 1000){  //①
                    totalPrice = 2000;
                    System.out.println("基本料金 2000 円です。");
                    continue; //②
          }
          totalPrice = packet*2;
          System.out.println("現在の利用料は"+totalPrice+"円です。");
}
System.out.println("今月の利用料は"+totalPrice+"円です。");
```

今回の主役たる continue が途中に含まれているのがわかると思います。さあ、結果はどうなったのか、見てみましょう。

```
基本料金 2000 円です。
基本料金 2000 円です。
中略
基本料金 2000 円です。
現在の利用料は 2364 円です。
現在の利用料は 2384 円です。
中略
現在の利用料は 4366 円です。
今月の利用料は 4366 円です。
```

というわけで、continue の効果はお分かりいただけたでしょうか？

まず、①で packet が 1000 未満かどうかを判断します。パケットが 1000 パケット未満であれば、{}の中を実行しますよね。その後、基本料金 2000 円であることを表示した後、continue 文に到達します。ここで、continue までいくと、③の for 文へ移動します。そして、i が 1 増加して繰り返し処理が行われます。

このように、continue があった場合、強制的に繰り返し処理に戻ることになります。

言葉で説明すると難しいですが、図で見れば簡単です。

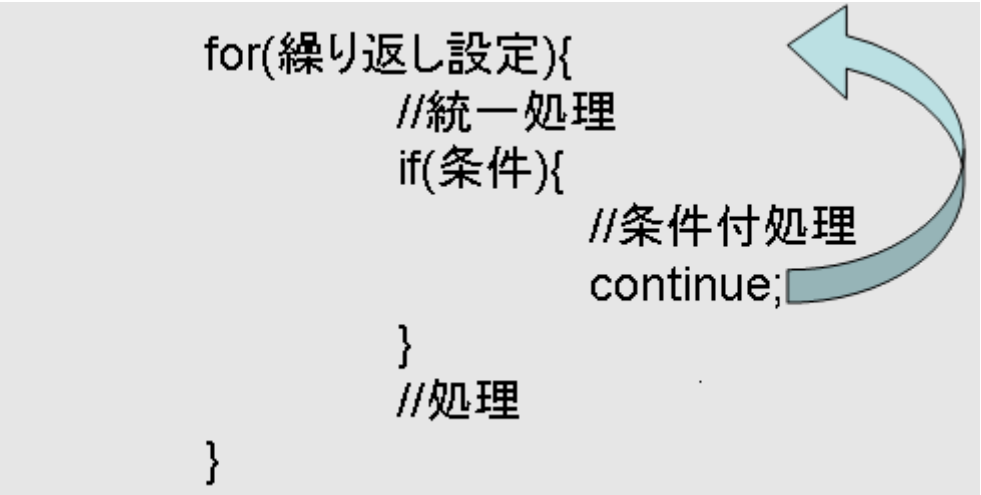

図 continue

このように、繰り返し処理の中のどこにいても、一気に繰り返し処理へ戻ることができるのが continue なのです。

利用可能なのは、for,while,do-while 文すべてです。

5.5.4 いきなり終了 break

さて、続いて break の説明です。ここでは、先ほどのパケット計算式にさらに以下の条件を追加したものを考えます。これも if 文で出てきたものですよね。

1、 1000 パケットまでは一律 2000 円

2、 1000 パケット以上なら，1 パケットに付き 2 円

3、 2000 パケット以上なら一律 4000 円

これを実現するために、先ほどのプログラムをこのように変更します。

```
for(int i = 0; i < packetPriceArray.length; i++){
        packet += packetPriceArray[i];
        if(packet < 1000){
                totalPrice = 2000;
                System.out.println("基本料金 2000 円です。");
                continue;
        }

        if(packet > 2000){
                totalPrice = 5000;
                System.out.println("2000 パケットを超えました");
                break;
        }

        totalPrice = packet*2;
        System.out.println("現在の利用料は"+totalPrice+"円です。");
```

これを実行してみると、このようになります。

```
基本料金 2000 円です。
基本料金 2000 円です。
中略
```

```
現在の利用料は 3956 円です。
2000 パケットを超えました
今月の利用料は 4000 円です。
```

現在の利用料金が 3956 円の次に、2000 パケット超えたことを知らせた直後に、いきなり最後に表示する今月の利用料が表示されています。まだ 30 日分データを表示していないのに、終了になってしまいました。

もう、大体お分かりいただけたかと思いますが、break は、繰り返し処理を強制終了する制御です。ここでは、packet が 2000 を越えたら、break に到達して for 文を終了するわけです。

図で見たほうが分かりやすいかもしれませんね。

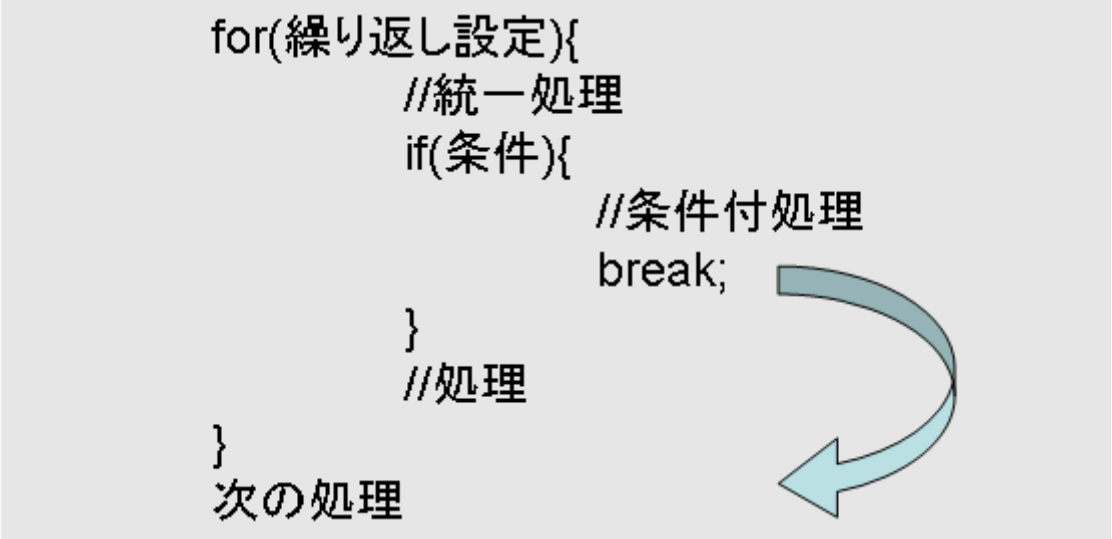

図　break 処理

というわけで、どうしても繰り返し処理がこれ以上続けたくなくなった場合は break を使うことにしましょう。

ちなみに、break を使うと強制的な終了ができますので、while 文などの条件式を複数つけることができますよ。

まあ、最初のうちは continue/break 共にあんまり使うことはないと思います。ですが、while でご紹介した**無限ループ**などで、break をうまく使えば無限ループを終了さえることができますので、活用してみてください。

また、for も while も条件式はひとつの()内に書かなければいけません。しかしながら、ものすごい複雑な条件を終了条件にしたい場合というのがあります。そのような場合、ループ自体は無限ループにしておいて、break で無理やり終了させるというのもひとつのテクニックです。

5.6 間違えやすい制御構文

5.6.1　　間違えやすい if の順番

さて，携帯電話のパケット料金のお話です.

2000 パケット以上だと一律 4000 円だけど，10000 パケットを超えたら追加料金が発生して 10000 円になるシステムを作ろうとした場合,

```
int packet = 12000;
int price = 0;
if(packet < 1000){
        price = 2000;
}
else if(packet > 2000){
        price = 4000;
}
else if(packet > 10000){
        price = 10000;
}
else{
        price = packet*2;
}

System.out.println("パケット代は"+price+"円です");
```

と書けば良さそうです.

しかし，これを実行すると

```
パケット代は 4000 円です
```

となり，10000 円を請求できなくなってしまいます. これではヘビーユーザがいた場合，携帯会社は大損です.

if～if else～else 文は必ず上から順番に処理されていくことを忘れていたためにこのようなことが発生してしまったんですね.

このコードをよく見てください.

```
else if(packet > 2000){//・・・①
        price = 4000;
}
else if(packet > 10000){ //・・・②
        price = 10000;
}
```

ここに注目すると，まず先に①でパケット代が 2000 円以上かどうかを確認しています. パケット数が 12000 ですので，当然ここで条件が満たされますので，price=4000 の行が実行され，この if 文から抜け出してしまいます.

つまり，パケット数が 10000 以上かどうか判定する②まで到達しないわけです.

このように，if 文を書くときは if-else if を各順番に気をつけないと絶対に到達しない場所が存在してしまうことになります.

こうならないように，厳しい条件を先に書く必要があります. 10000 パケット以上という条件は 2000 パケット以上という条件に含まれているので， こちらの方が厳しい条件だ

ということが出来ます．したがって，その条件を先に書くことにします．

先ほどの例を修正すると，

```
int packet = 12000;
int price = 0;
if(packet < 1000){
        price = 2000;
}
else if(packet > 10000){
        price = 10000;
}
else if(packet > 2000){
        price = 4000;
}
else{
        price = packet*2;
}

System.out.println("パケット代は"+price+"円です");
```

となります．これを実行すると．

```
パケット代は 10000 円です
```

ちゃんとヘビーユーザからパケット代を大量請求することができるようになりました．携帯会社社はウハウハ．ヘビーユーザはがっくしです．

というわけで,if-else if を書くときは**厳しい条件から先に書く**クセを付けておくようにしましょう．else 文はどうせ最後にしか着かないのであんまり深く考えなくてもいいんですけどね．

5.6.2 ついつい忘れる switch の break

switch 文では、整数の値によって行う処理を変更できるのですが、時々とんでもないミスをやらかしてしまうことが多々あります。そんな悲惨なパターンをここでお見せしましょう。

今回は、電話を市内、県内外、海外それぞれにかけた場合一分間にいくらかかるかを調べるプログラムを作って見ましょう。

ここでは、電話先を place 変数で処理し、0 なら市内、1 なら市外、2 なら県外、3 なら海外とします。

プログラムはこんな感じになりました。

```
int place = 1; //0:市内 1:県内 2:県外 そのほか:海外

int price = 0;
switch(place){//①
case 0: //②
        price = 10;// ③
case 1: //④
        price = 20; //⑤
case 2:
        price = 30;
default:
        price = 100; //⑥
```

```
        }

        System.out.println("通話料は3分で"+price+"円です");
```

さて、これを実行してみましょうか。今回は place が 0 なので、市内あての電話ということで、3 分 10 円という情報が帰ってくるはずです。

```
通話料は3分で100円です
```

おおっと。3 分でえらい高い値段をとられる電話になってしまいました。これじゃあ、ご近所さんと長電話するのもままなりません。お母さんもがっかり。

さて、なぜこんなことが起きてしまったのでしょうか？

本節のタイトルを見落としたうっかりさんじゃなければすぐに分かると思いますが、この switch 文、break 文が抜けているのです。

Switch 文には、break 文がないと case 文を突き抜けて次の行へといってしまうという決まりがあるのです。

つまり、switch 文①から②において case 0 にひっかかったので、③で price=0 となります。ここで switch 文は終わってほしいのですが、break 文を書き忘れていたので、そのまま④の行へ進んでしまいます。このとき、プログラムは④では何もすることがないので、そのままスルーして⑤へ移動します。ここで、price が 20 に上書きされてしまいます。

同様に、どんどん進んでいって最後には⑥で price が 100 になって swtich 文を抜けてくれることになります。

このように、switch において break 文を付け忘れるととんでもない結果になることが多々ありますので、必ず break を付け忘れないようにご注意ください。

ちなみに、わざと break を使わずに、複数の条件で同じことを書くという方法もあります。

```
        int place = 1; //0:市内 1:県内 2:県外 そのほか:海外

        int price = 0;
        switch(place){
        case 0:
        case 1:
        case 2:
                price = 20;
                break;
        default:
                price = 100; //⑥
        }
```

このようにプログラムを書くと、日本国内どこでも電話料金一律 3 分 20 円という料金体系を表すことができるようになります。ただし、値を決めた後は break を忘れないようにしましょう。